|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 3 |

**Название:**

Алгоритмы сортировки

**Дисциплина:** Анализ Алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-52Б |  |  | Н.А. Гарасев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Л.Л. Волкова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

Оглавление

[Введение 3](#_Toc52317386)

[1. Аналитическая часть 4](#_Toc52317389)

[2. Конструкторская часть 6](#_Toc52317390)

[2.1. Схемы алгоритмов 6](#_Toc52317391)

[3. Технологическая часть 10](#_Toc52317392)

[3.1. Реализация алгоритмов 10](#_Toc52317393)

[3.2. Тестирование функций 12](#_Toc52317394)

[4. Экспериментальная часть 13](#_Toc52317395)

[4.1. Интерфейс 13](#_Toc52317396)

[4.2. Сравнение алгоритмов по времени работы реализаций 15](#_Toc52317397)

[Заключение 21](#_Toc52317398)

[Список литературы 22](#_Toc52317399)

# Введение

Цель лабораторной работы: изучить и применить на практике алгоритмы сортировки массивов, а также проанализировать трудоемкость этих алгоритмов и затрачиваемые ресурсы. В данной лабораторной работе рассматривается сортировка пузырьком, сортировка расческой и сортировка выбором.

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. Сортировка элементов достаточно распространенная проблема, которая встречается практически в каждой отрасли современного мира.

## В ходе выполнения лабораторной работы требуется решить следующие задачи.

* 1. Изучить выбранные алгоритмы сортировки.
  2. Реализовать алгоритм сортировки пузырьком.
  3. Реализовать алгоритм сортировки выбором.
  4. Реализовать алгоритм сортировки расческой.
  5. Дать теоретическую оценку всем вышеупомянутым алгоритмам.
  6. Сравнить алгоритмы по затраченным ресурсам.
  7. Проанализировать полученные результаты и сделать вывод о эффективности реализованных алгоритмов.

# Аналитическая часть

Сортировка массивов — одна из основных операций над массивами.

## Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются **N-1** раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

## Сортировка выбором

Шаги алгоритма:

* находим номер минимального значения в текущем списке
* производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
* теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

## Сортировка расческой

Основная идея «расчёски» в том, чтобы первоначально брать достаточно большое расстояние между сравниваемыми элементами и по мере упорядочивания массива сужать это расстояние вплоть до минимального. Таким образом, мы как бы причёсываем массив, постепенно разглаживая на всё более аккуратные пряди. Первоначальный разрыв между сравниваемыми элементами лучше брать с учётом специальной величины, называемой фактором уменьшения, оптимальное значение которой равно примерно 1,247. Сначала расстояние между элементами равно размеру массива, разделённого на фактор уменьшения (результат округляется до ближайшего целого). Затем, пройдя массив с этим шагом, необходимо поделить шаг на фактор уменьшения и пройти по списку вновь. Так продолжается до тех пор, пока разность индексов не достигнет единицы. В этом случае массив досортировывается обычным пузырьком.

# Конструкторская часть

На вход алгоритмы принимают массив и его длину. На выходе выдают отсортированный по возрастанию массив.

# Схемы алгоритмов

На рис. 2-4 приведены схемы алгоритмов:

1. сортировка пузырьком;
2. сортировка выбором;
3. сортировка расческой.

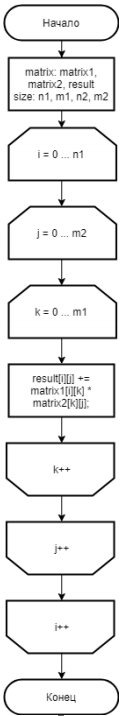


Рисунок 2. *Стандартный алгоритм перемножения матриц*

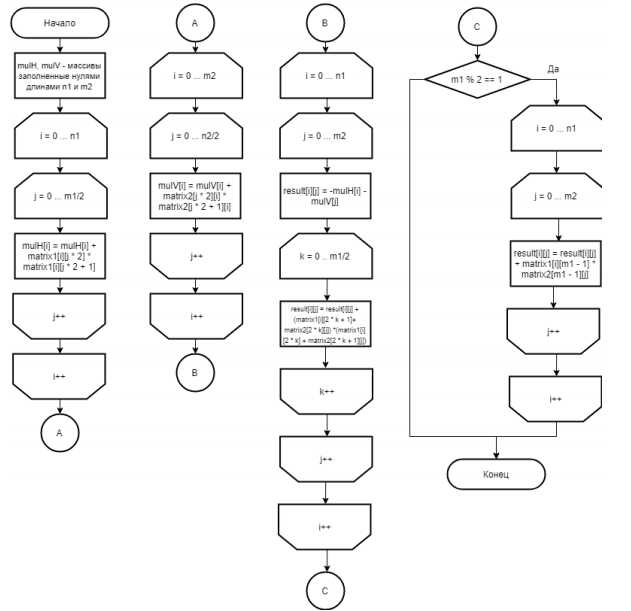


Рисунок 3. *Алгоритм Винограда*

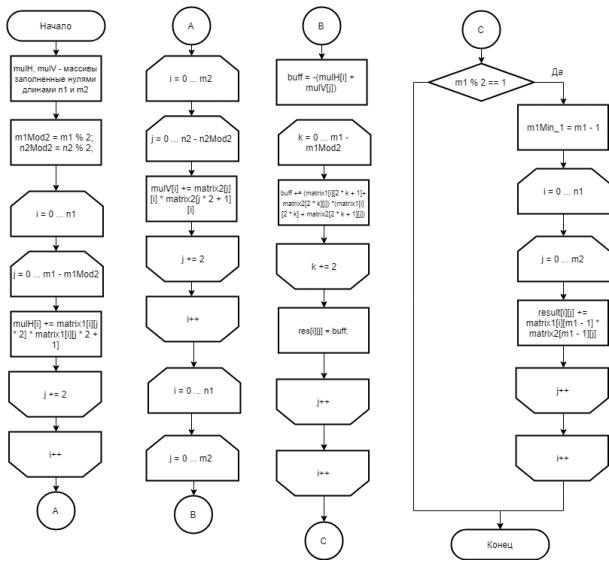
**

Рисунок 4. *Модифицированный алгоритм Винограда*

# Технологическая часть

В качестве языка программирования был выбран python, т.к. данный язык программирования имеет большое количество полезных библиотек для различных необходимостей, а также язык предоставляет средства для быстрого прототипирования и динамической семантики. Для замера процессорного времени была использована функция process\_time(), стандартной библиотеки python – time [2].

## Реализация алгоритмов

В листингах 1-4 представлена реализация алгоритмов сортировки массивов. В листинге 5 представлена функция для замера времени выполнения заданной функции на заданном количестве итераций на матрицах указанного размера.

Листинг 1. Реализация сортировки пузырьком

def bubble(arr, n):  
 i = 0  
 while i < n - 1:  
 j = i + 1  
 while j < n:  
 if arr[i] > arr[j]:  
 tmp = arr[i]  
 arr[i] = arr[j]  
 arr[j] = tmp  
 j += 1  
 i += 1

Листинг 2. Реализация сортировки выбором

def selection(arr, n):  
 i = 0  
 while i < n - 1:  
 j = i + 1  
 mina = arr[i]  
 mini = i  
 while j < n:  
 if arr[j] < mina:  
 mina = arr[j]  
 mini = j  
 j += 1  
 tmp = arr[i]  
 arr[i] = arr[mini]  
 arr[mini] = tmp  
 i += 1  
 return arr

Листинг 3. Реализация сортировки расческой

def comb(arr, n):  
 gap = (n \* 10 // 13)  
  
 while gap:  
 if 8 < gap < 11:  
 gap = 11  
 swapped = False  
 i = 0  
 m = n - gap  
 while i < m:  
 if arr[i + gap] < arr[i]:  
 tmp = arr[i]  
 arr[i] = arr[i + gap]  
 arr[i + gap] = tmp  
 swapped = True  
 i += 1  
 gap = (gap \* 10 // 13) or swapped  
 return arr

Листинг 4. Функция подсчета среднего времени выполнения алгоритма сортировки для случайного массива длиной size.

def time\_analyze(function, iterations=100, size=5):  
 t1 = process\_time()  
 for \_ in range(iterations):  
 start(size, function, False)  
 t2 = process\_time()  
 return (t2 - t1) / iterations

## Тестирование функций

Для модульного тестирования реализованных алгоритмах (см. листинги 1-3) была использована стандартная библиотека языка python – unittest. Все функции протестированы на различных входных данных.

Листинг 5. Функция сравнения матриц

**def** comp\_matr**(**matr1**,** matr2**):**

**if** **len(**matr1**)** **!=** **len(**matr2**):**

**return** **False**

**if** **len(**matr1**)** **==** 0 **or** **len(**matr1**[**0**])** **!=** **len(**matr2**[**0**]):**

**return** **False**

**for** i **in** **range(len(**matr1**)):**

**for** j **in** **range(len(**matr1**[**i**])):**

**if** matr1**[**i**][**j**]** **!=** matr2**[**i**][**j**]:**

**return** **False**

**return** **True**

Все тесты были пройдены успешно.

# Экспериментальная часть

При запуске программы первое, что видит пользователь – это интерфейс.

## Интерфейс

На рис. 5 представлено главное меню программы. В зависимости от выбранного пункта меню запускается соответствующий алгоритм

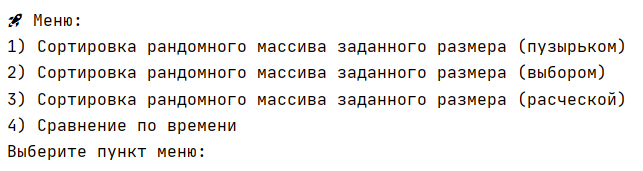


Рисунок 5. Интерфейс – меню

На рис. 6-9 приведены примеры работы программы при выборе пунктов меню 1-4.

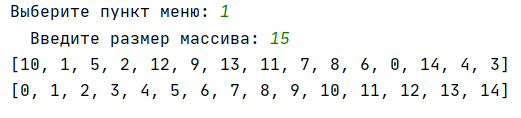


Рисунок 6. Пример работы программы при выборе пункта 1

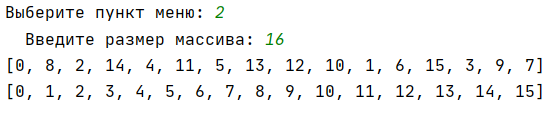


Рисунок 7. Пример работы программы при выборе пункта 2

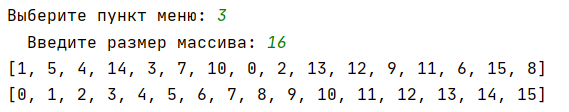


Рисунок 8. Пример работы программы при выборе пункта 3

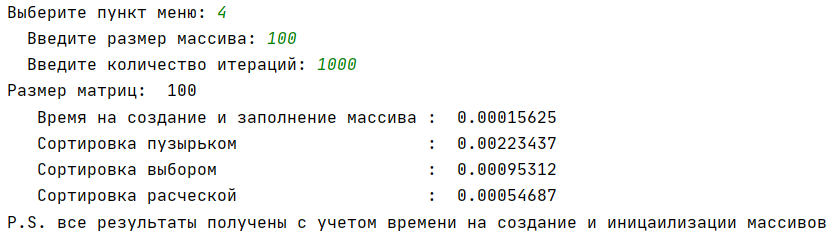


Рисунок 9. Пример работы программы при выборе пункта 4

## Сравнение алгоритмов по времени работы реализаций

Для сравнения в программе необходимо провести замеры процессорного времени для выполнения n – количества вычислений для матриц одинакового размера.

Таблица 3. Сравнение алгоритмов по времени

|  |  |  |
| --- | --- | --- |
| Алгоритм | Размер матриц | Среднее время, сек |
| Стандартный алгоритм | 10 | 0.00136250 |
| Алгоритм Винограда | 10 | 0.00155625 |
| Модифицированный алгоритм Винограда | 10 | 0.00119062 |
| Стандартный алгоритм | 25 | 0.02106250 |
| Алгоритм Винограда | 25 | 0.02340625 |
| Модифицированный алгоритм Винограда | 25 | 0.01775000 |
| Стандартный алгоритм | 50 | 0.16750000 |
| Алгоритм Винограда | 50 | 0.17093750 |
| Модифицированный алгоритм Винограда | 50 | 0.12218750 |

Замеры времени усреднялись для каждого набора экспериментов. Для этого все вычисления производились на случайных матрицах при 100-10000 итераций. Количество итераций обратно пропорционально размеру матриц. Это связано с тем, что время умножения больших матриц велико.

Таблица 4. Сравнение времени на малых размерах

|  |  |  |
| --- | --- | --- |
| Алгоритм | Размер матриц | Ср. время, сек |
| Стандартный алгоритм | 2 | 0.00001297 |
| Модифицированный алгоритм Винограда | 2 | 0.00001859 |
| Стандартный алгоритм | 3 | 0.00003906 |
| Модифицированный алгоритм Винограда | 3 | 0.00004844 |
| Стандартный алгоритм | 4 | 0.00009531 |
| Модифицированный алгоритм Винограда | 4 | 0.00010312 |
| Стандартный алгоритм | 5 | 0.00017656 |
| Модифицированный алгоритм Винограда | 5 | 0.00018906 |
| Стандартный алгоритм | 6 | 0.00030469 |
| Модифицированный алгоритм Винограда | 6 | 0.00030937 |
| Стандартный алгоритм | 7 | 0.00047813 |
| Модифицированный алгоритм Винограда | 7 | 0.00046406 |

**Вывод**: модифицированный алгоритм Винограда показывает наилучшее время на средних размерах, хотя обычный алгоритм Винограда показывает наихудшее время. Это связано с тем, что обычный алгоритм Винограда мало оптимизирован и приходится вычислять одни и те же значения несколько раз. Стандартный алгоритм работает быстрее всех на маленьких размерах. Выбирать алгоритм необходимо в зависимости от поставленной задачи. Если вам требуется быстро перемножать огромные матрицы, то модифицированный алгоритм Винограда справится с этой задачей лучше всех. Но если вам необходимо работать с матрица малых размеров (< 7), то стандартный алгоритм подойдет больше.

* 1. **Оценка трудоемкости алгоритмов**

Трудоемкость алгоритма — это количество операций, которые необходимо выполнить в процессе решения поставленной задачи с помощью этого алгоритма. Рассмотрим две трудоемкости для каждого из алгоритмов: наихудшая и наилучшая трудоемкость.

* + 1. **Сортировка пузырьком**

Рассмотри трудоемкость стандартного алгоритма сортировки – сортировка пузырьком

Наилучшая трудоемкость: 2 + (n - 1) \* (2 + 1 + 2 + (n - i - 1) \* (2 + 3))

Итог:

* + 1. **Сортировка выбором**

Рассмотри трудоемкость алгоритма Винограда для умножения матриц

Таблица 1. Трудоемкость алгоритма Винограда

|  |  |
| --- | --- |
| Части алгоритма | Трудоемкость |
| Инициализация mulH и mulV | 2 \* 3 |
| Заполнение mulH | 2 + n \* (2 + 2 + m / 2 \* (3 + 6 + 6)) |
| Заполнение mulV | 2 + k \* (2 + 2 + m / 2 \* (3 + 6 + 6)) |
| Подсчет результата | 2 + n \* (2 + 2 + k \* (2 + 7 + 2 + m / 2 \* (3 + 23))) |
| Условие нечетности m | 2 |
| Для матриц с нечетным m | 2 + n \*(2 + 2 + k \* (2 + 8 + 5)) |

Итог: 13 \* m \* n \* k + 7.5 \* m \* n + 7.5 \* k \* m + 11 \* k \* n + 8 \* n + 4 \* k + 14 +

+, где n, k, m – константы для матриц **A** с размерами n \* m и **B** с размерами m \* k. Выражение в [] принимает значение сверху, если m четное, а нижнее — иначе.

* + 1. **Сортировка расческой**

Рассмотри трудоемкость модифицированного алгоритма Винограда для умножения матриц

Таблица 2. Трудоемкость модифицированного алгоритма Винограда

|  |  |
| --- | --- |
| Части алгоритма | Трудоемкость |
| Инициализация mulH и mulV | 2 \* 3 |
| Переменная tmp | 3 |
| Заполнение mulH | 2 + n \* (2 + 2 + m / 2 \* (2 + 5 + 3)) |
| Заполнение mulV | 2 + k \* (2 + 2 + m / 2 \* (2 + 5 + 3)) |
| Подсчет результата | 2 + n \* (2 + 2 + k \* (2 + 5 + 3 + 2 + m / 2 \* (2 + 14))) |
| Условие нечетности m | 2 |
| Для матриц с нечетным m | 2 + 2 + n \* (2 + 2 + k \* (2 + 6 + 2)) |

Итог: 8 \* m \* n \* k + 5 \* m \* n + 5 \* k \* m + 12 \* k \* n + 8 \* n + 4 \* k + 17 +

+, где n, k, m – константы для матриц **A** с размерами n \* m и **B** с размерами m \* k. Выражение в [] принимает значение сверху, если m четное, а нижнее — иначе.

* 1. **Вывод**

Можно представить графики зависимости трудоемкости алгоритмов для умножения квадратных матриц с параметром x. На рис. 10 – 11 представлены графики таких функций.

1. Синий график – алгоритм Винограда
2. Зеленый график – модифицированный алгоритм Винограда
3. Красный график – стандартный алгоритм

****

Рисунок 10. *Графики трех алгоритмов по умножению матриц наилучший случай*

1. Фиолетовый график – алгоритм Винограда
2. Черный график – модифицированный алгоритм Винограда
3. Красный график – стандартный алгоритм

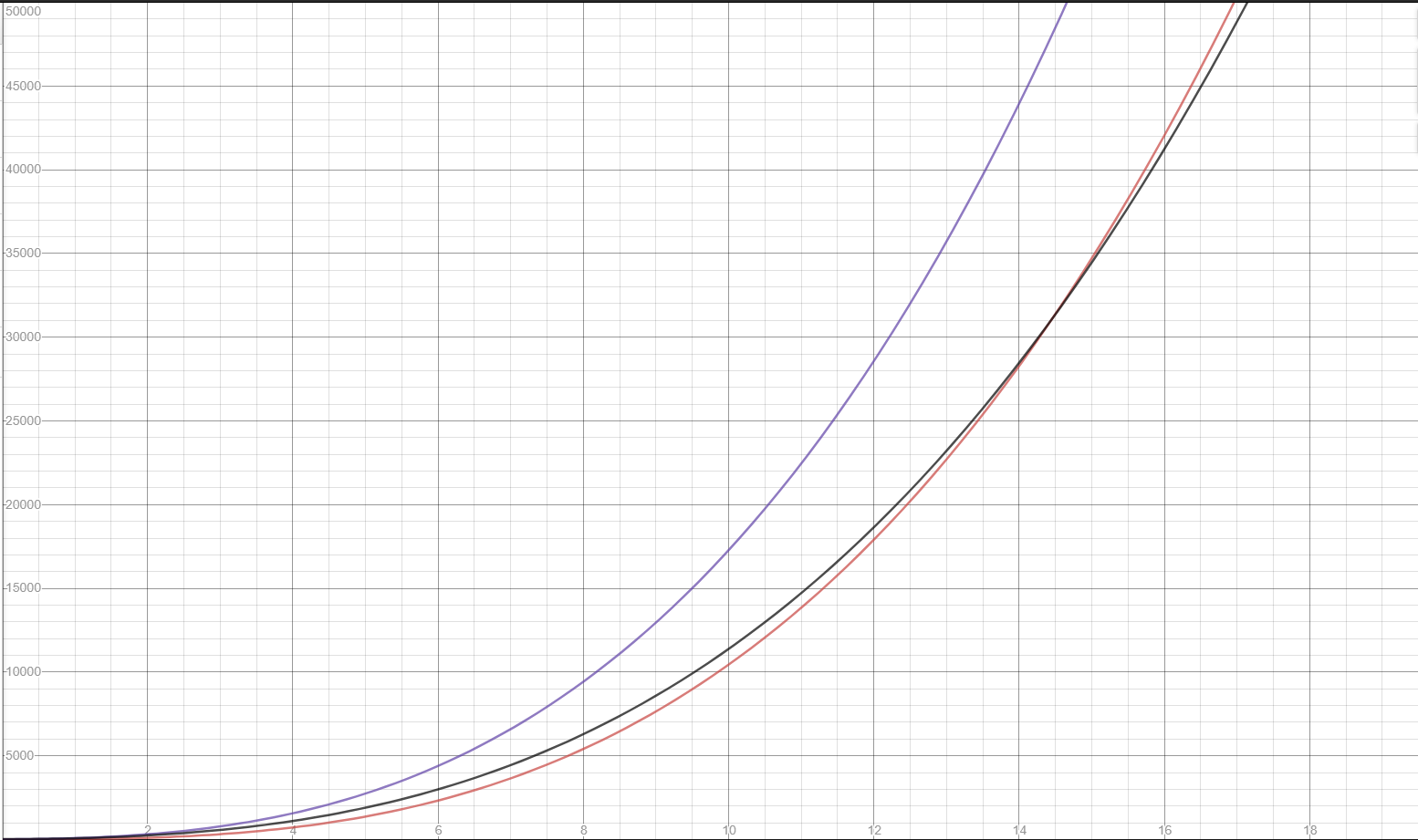
****

Рисунок 11. *Графики трех алгоритмов по умножению матриц наихудший случай*

**Вывод:** исходя из графиков, можно сделать вывод, что до определенного момента стандартный алгоритм умножения матриц наиболее быстрый, а также этот алгоритм стабилен, т.е. не зависит от четности m. Не модифицированный алгоритм Винограда показывает наихудшее время вне зависимости от четности. Модифицированный алгоритм показывает лучшее время с некоторого момента.

Заключение

В ходе работы были изучены алгоритмы умножения матриц. Реализованы 3 алгоритма, приведен программный код реализации алгоритмов по умножению матриц.

Была подсчитана трудоемкость каждого из алгоритмов. А также было проведено сравнение алгоритмов по времени и трудоемкости.

Цель работы достигнута. Получены практические навыки реализации алгоритмов Винограда и стандартного алгоритма, а также проведена исследовательская работа по оптимизации и вычислении трудоемкости алгоритмов.

Список литературы

1. Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход. –М.: Техносфера, 2017. – 267 c.

2. Официальный сайт Python, документация [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html>, свободный (дата обращения: 16.09.20).